**Topic Modeling**

**Introduction**

Another popular text analysis technique is called topic modeling. The ultimate goal of topic modeling is to find various topics that are present in your corpus. Each document in the corpus will be made up of at least one topic, if not multiple topics.

In this notebook, we will be covering the steps on how to do **Latent Dirichlet Allocation (LDA)**, which is one of many topic modeling techniques. It was specifically designed for text data.

To use a topic modeling technique, you need to provide (1) a document-term matrix and (2) the number of topics you would like the algorithm to pick up.

Once the topic modeling technique is applied, your job as a human is to interpret the results and see if the mix of words in each topic make sense. If they don't make sense, you can try changing up the number of topics, the terms in the document-term matrix, model parameters, or even try a different model.

**Topic Modeling - Attempt #1 (All Text)**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=b9bfd3055ba188c55824d99f971be318a2816eef&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f61646173686f66646174612f6e6c702d696e2d707974686f6e2d7475746f7269616c2f623962666433303535626131383863353538323464393966393731626533313861323831366565662f342d546f7069632d4d6f64656c696e672e6970796e62&nwo=adashofdata%2Fnlp-in-python-tutorial&path=4-Topic-Modeling.ipynb&repository_id=142180641&repository_type=Repository#Topic-Modeling---Attempt-#1-(All-Text))

*# Let's read in our document-term matrix*

**import** **pandas** **as** **pd**

**import** **pickle**

data = pd.read\_pickle('dtm\_stop.pkl')

data

*# Import the necessary modules for LDA with gensim*

*# Terminal / Anaconda Navigator: conda install -c conda-forge gensim*

**from** **gensim** **import** matutils, models

**import** **scipy.sparse**

*# import logging*

*# logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)*

*# One of the required inputs is a term-document matrix*

tdm = data.transpose()

tdm.head()

*# We're going to put the term-document matrix into a new gensim format, from df --> sparse matrix --> gensim corpus*

sparse\_counts = scipy.sparse.csr\_matrix(tdm)

corpus = matutils.Sparse2Corpus(sparse\_counts)

*# Gensim also requires dictionary of the all terms and their respective location in the term-document matrix*

cv = pickle.load(open("cv\_stop.pkl", "rb"))

id2word = dict((v, k) **for** k, v **in** cv.vocabulary\_.items())

Now that we have the corpus (term-document matrix) and id2word (dictionary of location: term), we need to specify two other parameters - the number of topics and the number of passes. Let's start the number of topics at 2, see if the results make sense, and increase the number from there.

*# Now that we have the corpus (term-document matrix) and id2word (dictionary of location: term),*

*# we need to specify two other parameters as well - the number of topics and the number of passes*

lda = models.LdaModel(corpus=corpus, id2word=id2word, num\_topics=2, passes=10)

lda.print\_topics()

*# LDA for num\_topics = 3* lda = models.LdaModel(corpus=corpus, id2word=id2word, num\_topics=3, passes=10) lda.print\_topics()

*# LDA for num\_topics = 4*

lda = models.LdaModel(corpus=corpus, id2word=id2word, num\_topics=4, passes=10)

lda.print\_topics()

These topics aren't looking too great. We've tried modifying our parameters. Let's try modifying our terms list as well.

## Topic Modeling - Attempt #2 (Nouns Only)

One popular trick is to look only at terms that are from one part of speech (only nouns, only adjectives, etc.).Check out the UPenn tag set: <https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html>.

*# Let's create a function to pull out nouns from a string of text*

**from** **nltk** **import** word\_tokenize, pos\_tag

**def** nouns(text):

*'''Given a string of text, tokenize the text and pull out only the nouns.'''*

is\_noun = **lambda** pos: pos[:2] == 'NN'

tokenized = word\_tokenize(text)

all\_nouns = [word **for** (word, pos) **in** pos\_tag(tokenized) **if** is\_noun(pos)]

**return** ' '.join(all\_nouns)

*# Read in the cleaned data, before the CountVectorizer step*

data\_clean = pd.read\_pickle('data\_clean.pkl')

data\_clean

*# Apply the nouns function to the transcripts to filter only on nouns*

data\_nouns = pd.DataFrame(data\_clean.transcript.apply(nouns))

data\_nouns

*# Create a new document-term matrix using only nouns*

**from** **sklearn.feature\_extraction** **import** text

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer

*# Re-add the additional stop words since we are recreating the document-term matrix*

add\_stop\_words = ['like', 'im', 'know', 'just', 'dont', 'thats', 'right', 'people',

'youre', 'got', 'gonna', 'time', 'think', 'yeah', 'said']

stop\_words = text.ENGLISH\_STOP\_WORDS.union(add\_stop\_words)

*# Recreate a document-term matrix with only nouns*

cvn = CountVectorizer(stop\_words=stop\_words)

data\_cvn = cvn.fit\_transform(data\_nouns.transcript)

data\_dtmn = pd.DataFrame(data\_cvn.toarray(), columns=cvn.get\_feature\_names())

data\_dtmn.index = data\_nouns.index

data\_dtmn

*# Create the gensim corpus*

corpusn = matutils.Sparse2Corpus(scipy.sparse.csr\_matrix(data\_dtmn.transpose()))

*# Create the vocabulary dictionary*

id2wordn = dict((v, k) **for** k, v **in** cvn.vocabulary\_.items())

*# Let's start with 2 topics*

ldan = models.LdaModel(corpus=corpusn, num\_topics=2, id2word=id2wordn, passes=10)

ldan.print\_topics()

*# Let's try topics = 3*

ldan = models.LdaModel(corpus=corpusn, num\_topics=3, id2word=id2wordn, passes=10)

ldan.print\_topics()

*# Let's try 4 topics* ldan = models.LdaModel(corpus=corpusn, num\_topics=4, id2word=id2wordn, passes=10) ldan.print\_topics()

## Topic Modeling - Attempt #3 (Nouns and Adjectives)

*# Let's create a function to pull out nouns from a string of text*

**def** nouns\_adj(text):

*'''Given a string of text, tokenize the text and pull out only the nouns and adjectives.'''*

is\_noun\_adj = **lambda** pos: pos[:2] == 'NN' **or** pos[:2] == 'JJ'

tokenized = word\_tokenize(text)

nouns\_adj = [word **for** (word, pos) **in** pos\_tag(tokenized) **if** is\_noun\_adj(pos)]

**return** ' '.join(nouns\_adj)

*# Apply the nouns function to the transcripts to filter only on nouns* data\_nouns\_adj = pd.DataFrame(data\_clean.transcript.apply(nouns\_adj)) data\_nouns\_adj

*# Create a new document-term matrix using only nouns and adjectives, also remove common words with max\_df*

cvna = CountVectorizer(stop\_words=stop\_words, max\_df=.8)

data\_cvna = cvna.fit\_transform(data\_nouns\_adj.transcript)

data\_dtmna = pd.DataFrame(data\_cvna.toarray(), columns=cvna.get\_feature\_names())

data\_dtmna.index = data\_nouns\_adj.index

data\_dtmna

*# Create the gensim corpus* corpusna = matutils.Sparse2Corpus(scipy.sparse.csr\_matrix(data\_dtmna.transpose())) *# Create the vocabulary dictionary* id2wordna = dict((v, k) **for** k, v **in** cvna.vocabulary\_.items())

*# Let's start with 2 topics* ldana = models.LdaModel(corpus=corpusna, num\_topics=2, id2word=id2wordna, passes=10) ldana.print\_topics()

*# Let's try 3 topics* ldana = models.LdaModel(corpus=corpusna, num\_topics=3, id2word=id2wordna, passes=10) ldana.print\_topics()

*# Let's try 4 topics* ldana = models.LdaModel(corpus=corpusna, num\_topics=4, id2word=id2wordna, passes=10) ldana.print\_topics()

## Identify Topics in Each Document

Out of the 9 topic models we looked at, the nouns and adjectives, 4 topic one made the most sense. So let's pull that down here and run it through some more iterations to get more fine-tuned topics.

ldana = models.LdaModel(corpus=corpusna, num\_topics=4, id2word=id2wordna, passes=80)

ldana.print\_topics()

Output:

[(0,

'0.009\*"joke" + 0.005\*"mom" + 0.005\*"parents" + 0.004\*"hasan" + 0.004\*"jokes" + 0.004\*"anthony" + 0.003\*"nuts" + 0.003\*"dead" + 0.003\*"tit" + 0.003\*"twitter"'),

(1,

'0.005\*"mom" + 0.005\*"jenny" + 0.005\*"clinton" + 0.004\*"friend" + 0.004\*"parents" + 0.003\*"husband" + 0.003\*"cow" + 0.003\*"ok" + 0.003\*"wife" + 0.003\*"john"'),

(2,

'0.005\*"bo" + 0.005\*"gun" + 0.005\*"guns" + 0.005\*"repeat" + 0.004\*"um" + 0.004\*"ass" + 0.004\*"eye" + 0.004\*"contact" + 0.003\*"son" + 0.003\*"class"'),

(3,

'0.006\*"ahah" + 0.004\*"nigga" + 0.004\*"gay" + 0.003\*"dick" + 0.003\*"door" + 0.003\*"young" + 0.003\*"motherfucker" + 0.003\*"stupid" + 0.003\*"bitch" + 0.003\*"mad"')]

These four topics look pretty decent. Let's settle on these for now.

* Topic 0: mom, parents
* Topic 1: husband, wife
* Topic 2: guns
* Topic 3: profanity

*# Let's take a look at which topics each transcript contains*

corpus\_transformed = ldana[corpusna]

list(zip([a **for** [(a,b)] **in** corpus\_transformed], data\_dtmna.index))

Output:[(1, 'ali'),

(0, 'anthony'),

(2, 'bill'),

(2, 'bo'),

(3, 'dave'),

(0, 'hasan'),

(2, 'jim'),

(3, 'joe'),

(1, 'john'),

(0, 'louis'),

(1, 'mike'),

(0, 'ricky')]

For a first pass of LDA, these kind of make sense to me, so we'll call it a day for now.

* Topic 0: mom, parents [Anthony, Hasan, Louis, Ricky]
* Topic 1: husband, wife [Ali, John, Mike]
* Topic 2: guns [Bill, Bo, Jim]
* Topic 3: profanity [Dave, Joe]

**Additional Exercises**

1. Try further modifying the parameters of the topic models above and see if you can get better topics.
2. Create a new topic model that includes terms from a different [part of speech](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html) and see if you can get better topics.